Victorian 6502 User Group Newsletter

KAOS

For People Who Have Got Smart

Registered by Australia Post
Publication No. VBG4212

ADDRESS ALL CORRESPONDENCE TO 10 FORBES ST., ESSENDON, VIC. 3040

Vol.2 No.11 August 1982

Last week we received a letter from Kerry Lourash, who had some how managed to see a copy of our newsletter in America, and he made some complementary remarks about it. Kerry sent us a very interesting program which is on page 6 of this month's newsletter. He would like to correspond with fellow ML hackers with a view to trading tips and information. Kerry's address is:

Kerry Lourash

U.S.A.

In view of the interest being shown in the use of computers in education, and as Tasmania is the leading state in Australia, in their use, we asked KAOS member, Graeme Reardon, who is the senior Maths/Science master at Cressy District High School, to write an article describing the TASNET system. His article is on page 13.

As David Wilson will not be able to attend the next meeting, there will not be a Forth meeting, but David Anear has been busily writing Forth programs and will be demonstrating his version of BREAKFORTH and his new R.G.B. driver.

The next meeting will be held on Sunday 29th August at 2pm at the Essendon Primary School, corner of Raleigh St and Nicholson St, Essendon. As this meeting is during the school holidays, we are not sure how many school children will be attending the morning session, but feel certain the very keen ones will be there, so would the early arrivers please bring their computers along as usual.

****

INDEX

MORE MODIFICATIONS TO DISK BASIC

A few months ago, I showed how to create some free space within the BASIC interpreter. This month I will describe how I have used up some of this space. The routines described previously must have been implemented (unless you relocate the routines given here outside the areas normally used by DISK BASIC (0200 to 2300)).

One of my complaints regarding BASIC is the clumsiness of the facilities for interfacing with machine-code subroutines, especially in the area of passing parameters and returning results.

There are two basic methods:
POKE 8956, HI : POKE 8955, LO: Y=USR(X)
which allows a single integer parameter to be passed and a single integer result to be returned. The POKE addresses, of course, differ from ROM BASIC. It would make it too easy to transfer programs form ROM-BASIC to DISK-BASIC if they used the same addresses. The address of the routine must be known (in decimal) and the low and high parts calculated and POKEd.

The second method is specific to the disk system:
DISK!"GO 1234" which requires that the address be known in hexadecimal and does not permit any parameters to be passed or results to be returned.

In Disk BASIC, both the USR function and the DISK! operation cause pages 0 and 1 to be swapped (with 2F79 to 3178) producing an overhead of more than 21ms (at 1MHz) which removes some of the advantages of machine coded routines for high-speed applications. (The page swap can be by-passed in the case of the USR function. This strategy is not advised for the DISK! function.)

To overcome these problems, I 'invented' a CALL statement which replaces the NULL statement, a statement that is hardly ever used and can 'easily' be replaced. NULLX is equivalent to POKE21,X on a disk-system (POKE13,X on a ROM system).

There are two forms of the CALL statement implemented. The address of the machine code routine may be given by either a hexadecimal constant or a numeric expression (sorry, no strings allowed).

Parameters are 'easily' passed and results returned by CALLing suitable routines within BASIC. To illustrate this, the code below includes two, CALLable routines.

CALL$17B6,P1,P2 (or CALL 6070,P1,P2) returns the address of P2 in P1. P1 should be a simple real variable. P2 may be an integer, string or real variable and may be simple or an array element.

CALL$186C,X prints X on the current default device in hexadecimal. X should be between 0 and 255. The address of the CALLed routine is assumed to be a normal BASIC expression unless it begins with a $ in which case it must contain 4 hex digits.

The modifications to BASIC are:

```
0222  D0 17    (CALL-1)      replaces address of NULL routine
02C5  43 41    'CA'          (replaces 'NU')
17B6  20130E   JSR  0E13     check for comma
17B9  202E0F   JSR  0F2E     get address of first parameter
17BC  8596     STA  96       store address
17BE  8497     STY  97
17C0  20130E   JSR  0E13     check for second comma
17C0  20130E   JSR  0F2E     get address of 2nd parameter
17C6  850A     STA  0A       swap A register and low in Y
17C9  A40A     LDY  0A
17CB  201812   JSR  1218     convert integer to real
17CE  4CCB1A   JMP  1ACB     store result into (96,97) and exit
CALL
17D1  C924     CMP  #24      compare with '$' hex address?
17D3  D010     BNE  17E5     no, decimal address instead
17D5  205519   JSR  1955     get two hex characters
17D8  851A     STA  1A       high byte of routine address
17DA  205519   JSR  1955     next two hex characters
17DD  8519     STA  19       low byte of address
```

```
17DF  20C000  JSR  00C0     get next character/token
17E2  6C1900  JMP  (0019)   jump to CALLed routine
17E5  4C6318  JMP  1863     get decimal address

1863  20B90C  JSR  0CB9     evaluate address of routine
1866  207216  JSR  1672     check range (0≤X≤65535) and store in 19,1A
1869  6C1900  JMP  (0019)   jump to CALLed routine
186C  206C16  JSR  166C     evaluate expression and check range (0≤X≤255)
186F  8A      TXA           transfer value to A
1870  4C992D  JMP  2D92     print in hex and exit

1955  206419  JSR  1964     convert hex digit to binary
1958  0A      ASL  A        shift
1959  0A      ASL  A        to
195A  0A      ASL  A        top
195B  0A      ASL  A        nibble
195C  8519    STA  19
195E  206419  JSR  1964     get 2nd hex digit
1961  0519    ORA  19       combine with first digit
1963  60      RTS           Return with result in A
1964  20C000  JSR  00C0     get next character
1967  2093FE  JSR  FE93     convert hex digit to binary using Monitor ROM
196A  10F7    BPL  1963     return if valid hex character
196C  4C1E0E  JMP  0E1E     invalid hex character, so report SN error
```

The following program demonstrates the use of the CALL statement with the two CALLable routines given above. The program takes a decimal value between 0 and 65535 and prints it out as a 4 character hexidecimal value.

```
  5 PRINT"convert decimal to hex"
 10 DISK!"ME 7000,7000"      set memory pointer to spare memory
 20 A$="ZZ":X=0              define variables
 30 CALL6070,X,A$            get location of string descriptor
 40 POKEX+1,0:POKEX+2,112    point A$ to $7000
 45 INPUT"Decimal";Z
 47 Z1=INT(Z/256)            get top byte
 48 Z=Z-Z1*256               get low byte
 50 DISK!"I0,10              set default output to memory
 60 CALL$186C,Z1             output top byte in hex (to memory)
 70 CALL$186C,Z              output low byte in hex
 80 DISK!"I0,02              set default output back to screen
 90 POKEX,4                  set LEN(A$) to 4
100 PRINT A$                 print result
110 GOTO 10                  reset pointers and get next value
```

This program is unnecessarily complicated in order to demonstrate the use of both CALLable routines. The program really only requires lines 45,47,48,60 and 70, which must make it the shortest BASIC program for the Challenger to do decimal to hex conversion.

Rodney Eisfelder

****

FOR SALE

Programmable Sound Generator Board with construction details, cost $15.00, or available as a kit or fully assembled and tested. Contact Ron Kerry at the KAOS meeting or phone or write to

****

Siemens Model 100, Friction Feed, Auto switch off mechanism, Tape Reperforator and Tape transmitter, attachments, governed to 75 BWDS. V.G.C. Price $285.00

W. Babb

Superboard

NEWSLETTER OF THE OHIO SUPERBOARD USER GROUP, 146 YORK ST.,NUNDAH, QLD. 4012.

Some letters I received about the Basic 4 Eprom offered last month showed that readers were unclear about just which features of the original Basic 4 were deleted to make way for the new routines. NO useful features have been deleted. The routines replace some obsolete code left over from the days when the Basic was in RAM, as well as some unnecessary text.

This program, to suit owners of Cegmon monitor, will enable the screen and window clear keys CTRL Z and CTRL SHIFT N, direct from the keyboard. The cursor control keys, CTRL J,K,L, and M are also enabled. Normally, these are ?CHR$(x) functions.

```
10 DATA 96,32,70,251,44,3,2,16,1,96,201,0,208,7,162,18,160,0
20 DATA 76,128,162,201,13,240,2,201,7,240,227,201,32,144,4
30 DATA 201,125,144,219,76,155,255
40 FOR R=6150 TO 6189:READ Z:POKE R,Z:NEXT:POKE 536,7:POKE537,24
```

- -

From Bob Best comes this neat mod to fix that Break Key problem. First, buy an 18¢ rubber grommet Dia. 17MM, thickness 5MM, hole 8MM from your hardware shop. Gently pry up the Keycap, place the grommet over the barrel, and restore the cap. Now a brush or accidental keypress will have no effect. You will need a firm push to operate the reset function.

- -

Hardware Review - EPROM EXTENDER

Eprom Extender is a bare 48mm X 120mm single sided printed circuit board, etched and drilled to accomodate three Eproms and a LS138 decoder I.C.

You solder in a wirewrap socket, and the pins plug into the Monitor Rom socket in the Superboard, which must be configured for a 2716 Eprom. The Monitor Eprom then plugs into the wirewrap socket. Three other connections are made from the Eprom Extender board to the Superboard, for decoding purposes, and the board is then strapped to provide a range of addresses for the two 2716 or 2732 Eproms which fit in the remaining two places on the Extender.

The review board was very well made, with no inter-track shorts or breaks, and was coated with protective resin. The instructions explained the wiring needed reasonably well, though a diagram showing the location of the chips on the Superboard would assist the owners without layouts and circuit diagrams.
On the Eprom Extender, pin 18 is tied to Zero volts, so the chips run in the power-up mode and will operate at 2MHz.

To sum up:- A neat and simple way of obtaining an extra 4 or 8k of ROM.
The Eprom Extender is available from Bert Patterson,
Cost is $8.50 plus $1 for Pack & Post.

- -

From Peter Wiseman comes this simple routine,when a few quick decimal to hex conversions are needed.

```
10 DATA 32,8,180,132,90,133,89,76,0,254
20 FOR R=565 TO 574:READ Z:POKE R,Z:NEXT:POKE 11,53:POKE 12,2
```

Run the program, then put the number you want to be converted to hex in the argument of the USR function. eg X = USR(65025) The program will do the job, then jump to monitor mode. Address $0059 contains the high byte (FE)
Address $005A " " low " (01)

Superboard

Software Review - COMPACT

Compact is a machine code utility occupying $1E00 - $1FFF or $3E00 - $3FFF. Both versions are supplied on the tape.

As the name suggests, Compact is used to pack up Basic programs into the minimum number of lines, deleting lines that have been compacted, and no longer required. It does the job well, and quickly, a 5k Basic program compacting in under a minute. The only thing that I could get it to do wrong was to put two colons together on a line, which caused no problems in the subsequent running of the program.

The instructions are short, but adequate, giving directions on how to solve problems associated with Basic 5 statements in programs. The routine actually needs from $1C00 to run, and you have to manually protect it from being over-written by Basic before you load - something easy to forget. The program could have been made self-protecting by adding A9 00 85 85 A9 1C 85 86 4C 74 A2 before $1E00, and making the routine self start from $1DF5.

To sum up:- An excellent and useful utility.
Compact is available from Premier Publications, 208 Croydon Rd.,Anerley, London SE20 7YX , priced at 8 Pounds + postage. The OSUG Library has one for Library members who would like to assess it personally. 27¢ + 40¢ stamps + label please.

- -

Firmware Review - Character Sets

The product is an Eprom which can replace the OSI character generator, or be piggybacked onto it. This can give alternative character sets, either switch selectable or software selectable. The Eprom supplied can be a 2716 or a 2732. In both cases, modification of the computer board is very easy.

The following options are available:-

(1) Standard Set. Available as one half of a 2732 Eprom only. For those who don't like to piggyback, and want an original set.

(2) Enhanced Set. Basically similar to OSI, but some of the rarely used characters have been replaced. All letters and numbers have been shortened by one pixel, making them infinitely easier to read on the screen. Small letters have descenders, which also improves the presentation. Added characters include black and matching white chess pieces, helicopters, cars and aircraft in all directions, 3/4 blocks for double resolution graphing, a parachute, and random characters and others to produce explosions and smoke or dust effects. Available as a 2716 or part of a 2732.

(3) Hi-Res Set. As in Practical Electronics, April 1982, pages 41-44. These give 4x resolution and are especially suitable for 24 x 48 or 32 x 64 screens. Available as 2716 or part of a 2732.

By piggybacking a 2732 onto your OSI character generator, you can have three character sets available at the flick of a switch. Yet to come is a scientific set with gaming characters replaced by various symbols and subscripts etc. New graphics charts and full documentation are provided with each chip. 2716 costs $12.80 and 2732 costs $17.80 including packing and post. Send your details to Bernie Wills,

Ed Richardson.

OSI GOFAST

by Kerry Lourash

GOFAST is a one-page machine language program designed to speed up GOTO and GOSUB commands. It implements labeled GOTOs and GOSUBs that can be renumbered and edited. With GOFAST, BASIC lines are found from 25% to over 300% faster. If you have a BASIC-in-ROM C1P or an OSI computer with a CTRL C vector in RAM, you can use this program.

A Little Background:

When a GOTO command is executed in BASIC, this is what happens: The number of the target line is converted to a two-byte hex number. The high byte of this number is compared to the high byte of the current line number (the line that contains the GOTO). If the high byte of the target line is greater than the high byte of the current line number, BASIC starts searching for the line at the next line after the GOTO. Otherwise, the search starts at the beginning of the program. The search is done by comparing the target line number with the line number stored at the start of every line. If a comparison fails, BASIC finds the location of the next line by using the next-line pointer and does another comparison.

A LINE OF BASIC:

/ 00	/ 53	/ 03	/ 0A	/ 00	/ CODE
NULL	($0353) NEXT LINE POINTER		(10) LINE #		

When the target line is found, its address is loaded into the parser pointer ($C3,C4) and BASIC resumes program execution at the target line.

GOSUBs are handled a little differently. Firstly, the stack is checked to see if there is enough room to accomodate the information that must be stored. Next, the contents of the parser pointer ($C3, C4), the current line number ($87, 88), and a GOSUB token ($8C) are pushed onto the stack. Counting the return address to the BASIC execution loop (which was already on the stack), seven bytes are stored. Finally, BASIC calls the GOTO routine, which finds the target line and resumes program execution.

A Faster Way:

BASIC searches for the target line every time a GOTO or GOSUB is encountered. This method is fine for occasional use. When the search is repeated thousands of times in a loop, however, the program slows down dramatically. Why does BASIC use this method? My guess is that this way is most memory-efficient.

Suppose BASIC used two-letter labels for target lines, with the address of each label in an indexed table. The table would occupy 26X26X2 bytes of RAM! If the labels were stored in the order of their occurrence in the program, memory usage might be less than that of the indexed table. Retrieval would be slower, however, because a comparison search would be necessary to find the address!

So, what is left? POKEing the hex address of the target line number after the GOTO or GOSUB token would be both time and memory efficient. Unfortunately, the OSI ASC11 SAVE and LOAD routines couldn't handle those bytes. Converting the hex address to ASC11 and storing it after the token would be better, but both of these methods would make the program logic undecipherable for us poor humans.

The GOFAST Way:

GOFAST uses one-letter labels (A-Z). This strategy keeps the label table small. More letters may be used in a label, but only the first letter is read. For a GOTO, the command is: #TO A. I use #>A for a GOSUB, but the ">" can br replaced with a different label if you wish. The target line is labeled with: REM#A. This label is placed in the line preceding the target line. You may use a separate line for the label or combine it with another line.

Since there are only 26 possible labels, it is advisible to use labels only for GOTOs or GOSUBs that are called frequently.

While running a program, CTRL C is disabled. The CTRL C vector at $021C, 021D is changed to point to GOFAST. GOFAST contains a streamlined version of the BASIC execution loop that resides at $A502-A5FE.

A short BASIC program is used to load GOFAST. It lowers the top-of-RAM pointer (MEMTOP) one page to protect GOFAST and sets the USR vector to the initialization routine.

Here's a short program to test the speed of the GOFAST routine:

```
30 X=USR (X):REM CALL INIZ. ROUTINE
40 FOR I=I TO 10000
50 #TO LOOP: REM#LOOP
60 NEXT
```

Note that the target REM causes a jump to line 60, not line 50.
To convert the program to stock BASIC, change:

```
30 PRINT
50 GOTO 60
```

Put ten or twenty BASIC lines (1 PRINT, 2 PRINT, etc.) in line numbers 1-29. The stock program's execution time will increase 200-300%, but the GOFAST program will not slow down.

BASIC V.S. GOFAST:

Here is a list of the BASIC execution loop's functions:

1 CTRL C check
2 Copy parser pointer into CONT pointer ($8B, 8C).
3 Check for null (start of line).
4 Check for a colon (start of statement).
5 If null or colon not found, print "SN ERROR".
6 Check next-line pointer to see if program ends.
7 If no more lines, to immediate mode.
8 Put current line number into current line pointer ($87, 88).
9 Update parser pointer to start of line.
10 Call parser ($00BC) to get first character of line.
11 Call BASIC execution routine ($A5FF).
12 Loop to #1.

GOFAST's version of the loop eliminates #1, 2, and 5. The net result is that GOFAST can check for a "#" at the start of every statement and still run faster than stock BASIC! I wonder how much faster BASIC would run with most of the error checking removed? For those speed freaks among us, steps 6, 7, and half step 8 could be removed if the END command is used and there are no bugs in the program to be run.

See GOFAST Run:

GOFAST consists of three parts: First, a 52 byte table that holds the addresses of the 26 labels, located just above MEMTOP. Second, an initialization routine that is run at the start of the user's program. It searches the user's program for labels and stores their locations in the table. Third, an operation routine that executes the #TO # > commands.

Since the initialization routine fills the table with zeroes before it searches for labels, any call to a non-existent label will send the program to warmstart and the immediate mode.

Side Effects:

Every program has its weak points. GOFAST doesn't do a foolproof job of error checking when it loads addresses into the label table, or when it looks them up. As a result, addresses may be retrieved or stored in the GOFAST program area, if the correct command format is not used. The effects will be unpredictable and might crash the program. I highly recommend the standard practice of SAVEing a program at intermediate stages of development to be safe.

If a BASIC program is terminated with an END command, GOFAST will not reset the CTRL C vector. After running such a program, you will find that the LIST command doesn't work the first time it is used.

Next page please

GOFAST's END routine resets the CTRL C vector. C2P owners with a RAM CTRL C vector should change LDA #$9B to LDA #$F1. In the BASIC load program, change 155 to 241 in line 30250.

Many thanks to Earl "The Pearl" Morris for bug-catching and good advice.

```
.A

                        ;
                        ; BASIC GOFAST MOD.
                        ; BY KERRY LOURASH
                        ;
                        ; FORMAT: $> A  (GOSUB)
                        ;         $TO B (GOTO)
                        ; TARGET: REM$A
                        ;         REM$B
                        ;
                          CTRLC=$021C       CTRL C VECTOR
                          INDEX=$5E         TEMP. STORAGE FOR TABLE INDEX
                         MEMTOP=$85         TOP OF RAM (+1)
                         PARSER=$00BC       PARSER SUBROUTINE
                          POINT=$AA         POINTER FOR INIT. ROUTINE
                          START=$79         START OF BASIC WORKSPACE
                        ;
                         *=*+52             MAKE ROOM FOR TABLE
                        ;
                        ; INITIALIZATION ROUTINE
                        ;
0034 A992    VECTOR LDA #ENTER*256/256 CHANGE CTRL C VECTOR
0036 8D1C02         STA CTRLC
0039 A900           LDA #ENTER/256
003B 8D1D02         STA CTRLC+1
003E            ;
003E A033    ZROTBL LDY #51      ZERO LABEL TABLE
0040 A900           LDA #0
0042 9185    Z1     STA (MEMTOP),Y
0044 88             DEY
0045 10FB           BPL Z1
0047            ;
0047 A579    SEARCH LDA START    SEARCH FOR REM LABELS
0049 85AA           STA POINT
004B A57A           LDA START+1
004D 85AB           STA POINT+1
004F A003    NEXLIN LDY #3
0051 C8      NEXCHR INY
0052 B1AA           LDA (POINT),Y
0054 F02D           BEQ FIXLIN
0056 C98E           CMP #$8E     IS CHAR A REM?
0058 D0F7           BNE NEXCHR
005A C8      N1     INY
005B B1AA           LDA (POINT),Y
005D C923           CMP #'$      IS CHAR A "$" ?
005F 90F9           BCC N1       IS CHAR A SPACE?
0061 D020           BNE FIXLIN   IF $ NOT FOUND, TO NEXT LINE
0063 C8      N2     INY
0064 B1AA           LDA (POINT),Y  GET LABEL
0066 38             SEC
0067 E941           SBC #$41
0069 90F8           BCC N2       TRY AGAIN IF CHAR < $41
006B 0A             ASL A        DOUBLE THE INDEX
006C 855E           STA INDEX    SAVE THE INDEX
006E A000    STORE  LDY #0        GET ADDRESS OF NEXT LINE
0070 B1AA           LDA (POINT),Y
0072 E900           SBC #0       SUBTRACT ONE FROM LO BYTE
0074 AA             TAX
0075 C8             INY          GET HI BYTE OF ADDRESS
0076 B1AA           LDA (POINT),Y
0078 E900           SBC #0       DECREMENT IF NECESSARY
007A A45E           LDY INDEX    STORE ADDRESS IN TABLE
007C C8             INY
007D 9185           STA (MEMTOP),Y
007F 88             DEY
0080 8A             TXA
0081 9185           STA (MEMTOP),Y
0083            ;
0083 A000    FIXLIN LDY #0       SET POINT TO START OF LINE
0085 B1AA           LDA (POINT),Y GET LO BYTE OF ADDRESS
0087 AA             TAX
0088 C8             INY
0089 B1AA           LDA (POINT),Y GET HI BYTE OF ADDRESS
008B 86AA           STX POINT
008D 85AB           STA POINT+1
008F D0BE           BNE NEXLIN   IF HI BYTE <> 0, BRANCH
0091 60             RTS          RETURN TO BASIC
0092            ;
0092            ; OPERATION ROUTINE
0092            ;
0092 68      ENTER  PLA          PULL SUB CALL FROM STACK
0093 68             PLA
0094 A000           LDY #0       FIRST CHAR OF LINE A NULL?
0096 B1C3    LOOP   LDA ($C3),Y
0098 D01A           BNE COLON    NO, MUST BE A COLON
009A A002    NULL   LDY #2        CHECK HI BYTE OF NEXT LINE
009C B1C3           LDA  ($C3),Y ADDRESS
009E F051           BEQ END      END OF BASIC PROGRAM
00A0 C8             INY          SAVE CURRENT LINE NUMBER
00A1 B1C3           LDA ($C3),Y
00A3 8587           STA $87
00A5 C8             INY
00A6 B1C3           LDA ($C3),Y
00A8 8588           STA $88
00AA 98             TYA          MOVE PARSER POINTER TO START
00AB 18             CLC          OF CODE
00AC 65C3           ADC $C3
00AE 85C3           STA $C3
00B0 9002           BCC COLON
00B2 E6C4           INC $C4
00B4 20BC00  COLON  JSR PARSER   IS FIRST CHAR A "$" ?
00B7 C923           CMP #'$
00B9 F007           BEQ GOFAST   YES, TO GOFAST
00BB 20FFA5         JSR $A5FF    DO COMMAND
00BE A000           LDY #0
00C0 F0D4           BEQ LOOP     GO TO START OF EXEC. LOOP
00C2            ;
00C2 20BC00  GOFAST JSR PARSER   GET TOKEN, IGNORE SPACES
00C5 AA             TAX
00C6 20BC00         JSR PARSER   GET LABEL, IGNORE SPACES
00C9 E941           SBC #$41     CALCULATE INDEX
00CB 0A             ASL A
00CC A8             TAY
00CD E09D           CPX #$9D     IS TOKEN= "TO"?
00CF F015           BEQ GOTO
00D1            ;
00D1 A9A5    GOSUB  LDA #$A5     PUSH RETURN ADDRESS
00D3 48             PHA          ($A5C1)
00D4 A9C1           LDA #$C1
00D6 48             PHA
00D7 A5C4           LDA $C4      PUSH PARSER POINTER
00D9 48             PHA
00DA A5C3           LDA $C3
00DC 48             PHA
00DD A588           LDA $88      PUSH CURRENT LINE NUMBER
00DF 48             PHA
00E0 A587           LDA $87
00E2 48             PHA
00E3 A98C           LDA #$8C     PUSH GOSUB TOKEN
00E5 48             PHA
00E6            ;
00E6 B185    GOTO   LDA (MEMTOP),Y GET ADDRESS FROM TABLE
00E8 85C3           STA $C3        PUT IT IN PARSER POINTER
00EA C8             INY
00EB B185           LDA (MEMTOP),Y
00ED 85C4           STA $C4
00EF D0A9           BNE NULL     HI BYTE OF ADDRESS= ZERO?
00F1            ;
00F1 A99B    END    LDA #$9B     RESTORE CTRL C VECTOR
00F3 8D1C02         STA CTRLC
00F6 A2FF           LDX #$FF
00F8 8E1D02         STX CTRLC+1
00FB CA             DEX          RESET STACK
00FC 9A             TXS
00FD 4C74A2         JMP $A274    TO WARMSTART
```

```
LIST

30000 REM MOVE MEMTOP DOWN ONE PAGE
30010 A=PEEK(133):B=(PEEK(134)-1):POKE134,B
30020 REM CALCULATE START ADDRESS
30030 C=A+52+B*256
30040 REM POKE GOFAST INTO MEMORY
30050 FORI=0TO203:READ N:POKE C+I,N:NEXT
30060 REM CALC. ADDRESS OF ENTER ROUTINE
30070 REM AND POKE IT INTO INIT. ROUTINE
30080 E=A+146:IF E=>256 THEN E=E-256:H=1
30090 H=H+B:POKE C+1,E:POKEC+6,H
30100 REM SET USR VECTOR TO INIZ. ROUTINE
30110 E=A+52:IF E=>256 THEN E=E-256:B=B+1
30120 POKE 11,E:POKE 12,B
30130 DATA169,128,141,28,2,169,2,141,29,2,160,51,169,0,145
30140 DATA133,136,16,251,165,121,133,170,165,122,133,171,160,3,200
30150 DATA177,170,240,45,201,142,208,247,200,177,170,201,35,144,249
30160 DATA208,32,200,177,170,56,233,65,144,248,10,133,94,160,0
30170 DATA177,170,233,0,170,200,177,170,233,0,164,94,200,145,133
30180 DATA136,138,145,133,160,0,177,170,170,200,177,170,134,170,133
30190 DATA171,208,190,96,104,104,160,0,177,195,208,26,160,2,177
30200 DATA195,240,81,200,177,195,133,135,200,177,195,133,136,152,24
30210 DATA101,195,133,195,144,2,230,196,32,188,0,201,35,240,7
30220 DATA32,255,165,160,0,240,212,32,188,0,170,32,188,0,233
30230 DATA65,10,168,224,157,240,21,169,165,72,169,193,72,165,196
30240 DATA72,165,195,72,165,136,72,165,135,72,169,140,72,177,133
30250 DATA133,195,200,177,133,133,196,208,169,169,155,141,28,2,162
30260 DATA255,142,29,2,202,154,76,116,162
OK
```

ANOTHER R.G.B. DRIVER BOARD

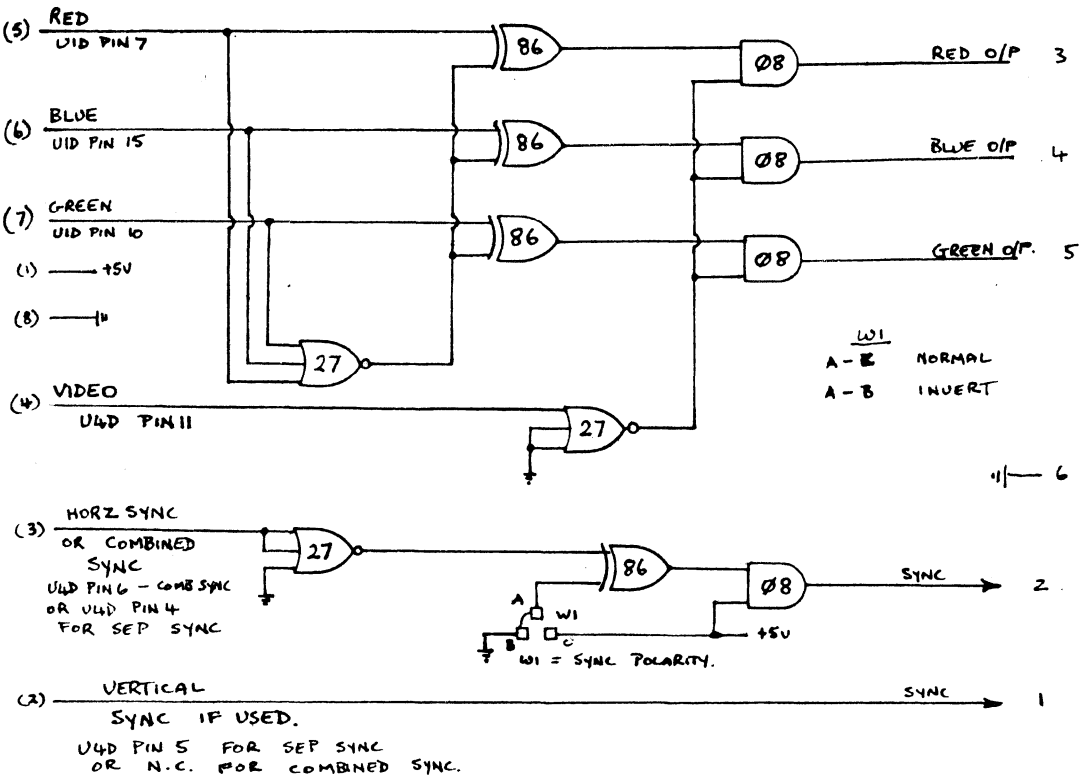
This little circuit has the following advantages over other designs:

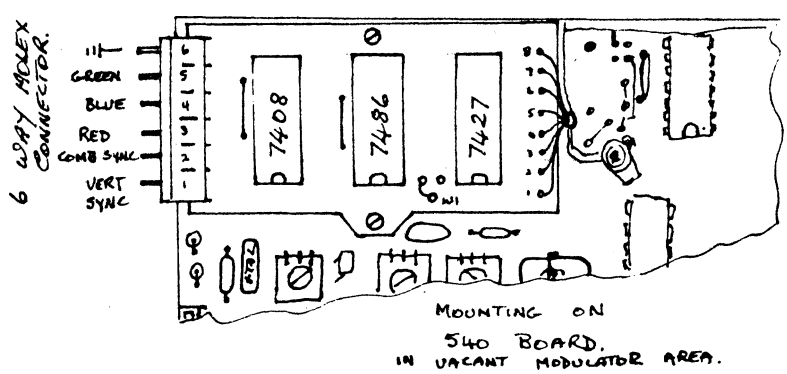
1. OHIO compatible colours
2. Simple circuitry
3. Can drive modified T.V. or RGB monitor (see Vol.2 No.2 page 2)
4. Combined sync or separate sync O/P
5. By changing O/P IC open collector or inverse O/P can be selected
6. W1 used for inverting sync with respect to O/P sync's
7. Can be interfaced to Superboard series II, 540 Board in C4, C8P systems and TASAN Video board.

The board layout shown is for the C4/C8 system.

The RGB Board plugs into the J5 connector on the TASAN Video board with similar connections for the Superboard.
If you are interested in the circuit boards, let me know and I will get some made.

David Anear





****

HOW TO COOK CHIPS BY BYTA BITROSE

Making unEproms into Eproms. Get out the frypan and set it on 175 C. (If cooking American chips, set it on 350 F). Put your unerasable chips in the pan and leave them for about 20 minutes. Do not baste them as this will make them greasy chips. Remove them and place them on a plate to cool before letting your eraser eat the bytes.

THE BEGINNING MACHINE LANGUAGE PROGRAMMER...part 2

This month, before looking at the instructions available on the 6502, we will coverthe stack and the status register in a little more detail.

The STACK is an area of memory organised on a LAST IN, FIRST OUT basis. The operation of placing information on the stack is termed a PUSH. A PULL operation retrieves the information. We can simulate the operation of the stack in BASIC by treating it as a dimensioned array.

```
DIM STack (255):SP=255:REM SP=stack pointer:A=accumulator
PUSH:ST(SP)=A:SP=SP-1
PULL:SP=SP+1:A=ST(SP)
```

If you follow this through you will see information is being 'stacked' just like a pile of plates. As information is placed on the stack, the stack pointer is automatically updated to point to the next position on the stack. Likewise, before a PULL occurs, the stack pointer is updated to point to the last entry placed on the stack. In this way, items added to the stack can be accessed sequentially, but in reverse order. The architecture of the 6502 is such that the stack is located at $0100-01FF (page 1).

The STATUS REGISTER comprises seven one bit 'flags' and an eighth bit which is not implemented. The flags are set to 1 for a true condition or cleared to zero for false. The flags are provided in order to:
(a) make decisions based on the contents of a register or memory location.
(b) indicate the current condition of the microprocessor.

The flags divide into two groups; four are known as status flags and the other three as condition flags. The STATUS flags are called carry (C), negative (0), zero (Z) and overflow (V).

In many operations, the CARRY can be considered to be a ninth bit on the accumulator. Its function is to signal when a carry or borrow is required as the result of an arithmatic operation.

The NEGATIVE (or sign) bit is set or cleared according to the state of the most significant bit after an operation on a register or memory. The CPU operates on the convention that a number is flagged as negative if the most significant bit is set, ie. signed binary arithmatic is assumed.

The ZERO flag is set to true (1) if the result of an arithmatic or boolean operation yielded an all zero result.

The OVERFLOW flag is seldom referenced in programming. It indicates whether the result of a twos' complement arithmatic operation is too big to be represented in one byte.

The CONDITION flags are called interrupt disable (1), break (B) and decimal mode (D).

The 6502 can operate in both binary and binary coded decimal (BCD) modes. In BCD mode, the maximum value that can be expressed in one byte is 99 (9X10+9 each represented in 4 bits). The carry is set if an operation causes a negitive to exceed this value. In this way, the CPU simulates decimal arithmatic. The DECIMAL flag indicates which mode is currently in operation.

Most microprocessors can be interrupted while they are in mid program and forced to jump to another program. This facility is used by peripherals to let the CPU know they need attention rather than have the CPU constantly checking whether a peripheral device is ready. The 6502 has three levels of interrupt:

non maskable
maskable
software

The BREAK key is an example of the way a non maskable interrupt (NMI) works; ie. it is always obeyed. The INTERRUPT DISABLE is the means used by the CPU to determine whether it is allowed to process a request by a peripheral for service. If interrupt disable is set the request will be ignored. Note that this flag is set by the CPU whenever any of the 3 types of interrupt occurs.

Next page please

The BREAK flag is set when a software interrupt (SWI or BREAK, the instruction not the key) is performed. Since NMI and SWI are treated the same way by the 6502, the programmer will decide what type of interrupt is being processed by examining the break flag.

The positions of the flags within the status register are:

Bit 0 C carry flag	Bit 4 B break flag
Bit 1 Z zero flag	Bit 5 unused (always 1)
Bit 2 I interrupt disable	Bit 6 V overflow flag
Bit 3 D decimal mode	Bit 7 N negative flag

The 6502 instructions can be divided into four distinct categories:

1. DATA TRANSFER instructions move data between two registers or between a register and memory. Some transfers involve arithmetic or logic operations. This group includes instructions such as LOAD, STORE, TRANSFER, ADD, SUBTRACT, AND, OR, EXCLUSIVE, OR (EOR or XOR) and COMPARE. PUSH and PULL are also included in this category.
2. OPERATIONAL instructions differ from the former group in that while the contents of a register changes as a result of the operation, no data transfer takes place. Operational instructions include INCREMENT, DECREMENT, CLEAR, SET, BIT, TEST, SHIFT, and ROTATE.
3. CONTROL and BRANCH instructions are of two types: condional and unconditional. Conditional branches test the state of one of the status registers and branch if the specified condition is met. The conditional branch works like the IF....THEN GOTO construction in BASIC. There are eight BRANCH instructions corresponding to each of the two states of the status flags. The unconditional branches include JUMP and BREAK.
4. SUBROUTINE LINKING gives the facility to call proedures located in another part of memory and to return to the same place where processing left off. These include JUMP to subroutine and RETURN from subroutine.

The 6502 offers 56 different instructions. To increase programming flexibility, there are 13 different methods of accessing information. Some instructions are available in as many as eight of these addressing modes.

In later articles, we will start to explore the various instructions and addressing modes.

David Dodds

****

DEAR PAUL,

Q. What mods are necessary to change the CPU speed from 1 MHz to 2 MHz?
A. I suggest that you look back to Vol.1 No.8 of the KAOS newsletter and on page 9 you will find full circuits for accomplishing this remarkable feat.

Q. What is the difference between standard monitor and extended monitor?
A. The standard, 65V monitor supplied with the Superboard etc. has very minimal machine code programming facilities. The extended monitor is designed to allow complex machine code operations, it allows you to display a block of memory, edit memory, disassemble memory (display memory by showing you the 6502 opcodes) set breakpoints (points in your machine code program that cause control to be transferred to the extended monitor). The extended monitor is available on cassette or in EPROM, and comes standard with OS65D disks.

Q. What are the specific functions of FRE(X) and NULLX?
A. FRE(X) is a function which returns the amount of free memory, in bytes, it is usually used in the form PRINT FRE(X) or Y=FRE(X). NULLX is a statement which causes X null characters (CHR$(0)) to be output after each carriage return, when data is being sent to the ACIA, ie. When you are saving a program to cassette. It is used when you have a device which takes a while to process a carriage return, eg. a teletype or a printer on the serial port.

Q. What is the @ command and how do you use it?
A. The @ command in BASIC simply deletes the current line.

THE TASAN VIDEO BOARD - SOME "FIXES"

This is probably going to be the last article that I shall write on the TASAN Video board, so I will devote it to some alterations (both to it, and to my programmable character generator - see Vol.2 No.9 newsletter).

Many of you who have built the Video board have complained about random characters occuring here and there on the screen. The first method I would suggest for fixing this problem is to replace IC 26 (a 74LS03) with a 74LS38. If this doesn't seem to fix the problem then you should piggy-back 74LS139 on to the 74LS139 on the board (IC 48) soldering pins 1, 2, 8 and 16; bend all other pins out a little so they don't make contact with the pins beneath them. Solder a length of wire wrap wire from pin 3 of your new chip to pin 12 or 13 of IC 27 (a 74LS20). Solder a length of wire wrap wire from pin 5 of your new chip to pin 10 of IC 17 (a video 2114); cut the track from pin 12 of IC 16 at the pin. This should fix the problem.

The cause of this problem is that when the computer is accessing the screen RAM, it is given priority over all the video signals, including the R/W line to the video RAMs, so, if you write data to the screen, the R/W line goes low to indicate writing; but when the computer has finished writing to the screen, priority over the video signals is given back to the scanning circuitry. Unfortunately, the 74LS157s which do this, appear to be a little slow and the R/W line may still be in the write state, which causes a character to appear at a random address on the screen.

The new video R/W circuitry looks like fig.1

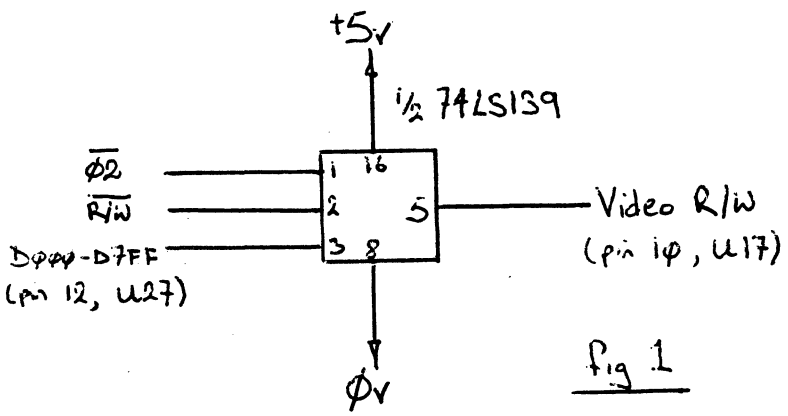


fig 1

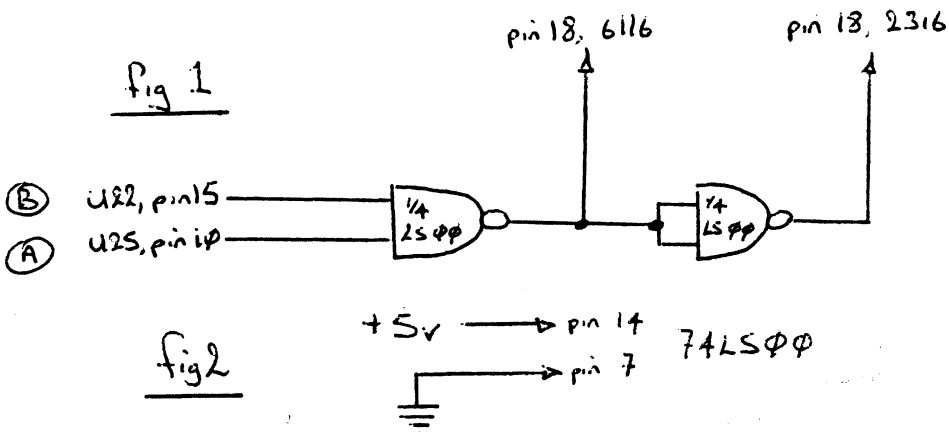


fig2

Some Video boards have problems where the characters in the 32 X 32 format "break up" or are otherwise "messy". This is easily cured by replacing U5 - a 74LS157 with a standard (not LS) 74157.

If your screen is plagued by black lines running through it - especially when colour is enabled. This is cured by replacing the 8T28s with faster ones, (just replace them, starting from the right hand end until the problem is fixed).

There are two problems with my Programmable charactor generator, firstly on my circuit diagram, I have labeled pin 20 of the 6116 and 2316 wrongly. Pin 20 of the 6116 should be grounded, and pin 20 of the 2316 should have the strap option, which I have shown coming from pin 20 of the 6116.

Secondly, the circuitry to pin 18 of the 6116 and 2316 is slightly wrong. The new circuitry should look like fig.2

Paul Dodd

TASNET

The Tasmanian Education Department Network (TASNET) through the Elizabeth Computer Centre (E.C.C.) services Schools, Colleges and Administrative Departments throughout Tasmania with a number of computers in various regions linked together by synchronous telephone line (4800 baud) or local asynchronous lines (1 megabaud in Hobart).

TASNET currently consists of one Digital VAX 11/780, three PDP 11/70s and two 11/34s at the following locations:

HOBART	:	1 only VAX 11/780	2 Mbyte
		2 only PDP 11/70	½ Mbyte each
LAUNCESTON	:	1 only PDP 11/70	½ Mbyte
DEVONPORT	:	1 only PDP 11/34	¼ Mbyte
BURNIE	:	1 only PDP 11/34	¼ Mbyte

There is approximately 2000 Mbytes of on-line storage available with off-line magnetic tape backup available at each computer - the whole system is backed up daily.

All High Schools and Colleges now have on-line facilities - usually by either 300 baud dedicated lines or 300 baud dial-up modems connected to V.D.U.s and/or printers (LA 34 or LA 36 usually) with on-line facilities for high speed line printers at each computer node. Many schools also have APPLE microcomputers with 48K memory and one disk drive. Software has been developed to allow down loading of Applesoft programs stored on the system and APPLES to be used as TASNET terminals.

Uses to which the system is put include:

1. Administration
2. Computer Studies courses at Years 9, 10, 11, 12
3. Computer Assisted Learning
4. Computer Awareness Courses at Years 7, 8.
5. Library Circulation System

In the Administration applications area, teacher support is offered through:

1. TASCIS an on-line cataloguing data base for school and college libraries - it now has over 100,000 books catalogued and library cards for these may be requested for printout on a special terminal.
2. THE TASMANIAN MEDIA CENTRE has over 10,000 items available for loan (films, tapes, videos, kits) and information on these items is available on-line to all schools (includes synopsis, suitability, availabilty) and schools can check their bookings (own only).
3. ABC REPORTER All schools TV and RADIO programs are catalogued on-line and times, suitability and synopsis for each is available as well as coming highlights and alterations.

Many schools have computerised their administration with various listings of classes, addresses, subjects and groupings, census informationetc. readily available.

The system has available BASIC, EXTENDED BASIC, PASCAL, COBOL, FORTRAN, LOGO, INTERFACING TO TURTLES AND A FILE MANAGEMENT PACKAGE. Computer assisted learning, games and other general purpose programs are available including assessment, survey and census data information.

Library circulation software has been developed in TASMANIA and is used for on-line circulation control in some branches of the State Library and in some schools and colleges.

APPLE microcomputers are supported and many programs have been written by E.C.C. staff and by teachers and are available for sale to interested groups and persons in other states. Very recently B.B.C. Protons have been recommended as the new supported computer for the next five years, although APPLES will continue to be supported. If enough interest is shown further information on APPLES could be made available in a later issue.

Graeme D. Reardon
Senior Master Maths/Science
Cressy District High School
Tasmania

THE DEBUGGER

The debugger is an extremely useful tool for persuading machine code routines to work. It allows the user to single step his program, returning control to a trace routine each time.

The key to the debugger lies in the hardware. In fact, all the debugger consists of is a 3 input NAND gate. (Makes you wonder why all micros don't have it.) The 6502 chip outputs a pulse on the SYNC pin each time it begins an instruction. If this is fed back to the NMI input, an interrupt would be generated each time an instruction is executed. Clearly, this would be useless as the NMI routine would also be traced and the system would wind itself up in knots. However, if this signal was passed through a NAND gate, (the SYNC output sends a positive going pulse, NMI is active low) the chip select line from the trace routine ROM could disable the debugger while executing the trace program. The third input of the NAND gate is simply connected to a flip-flop which is toggled by the 'DBG ON' and 'OFF' switches on the SYM keypad.

The trace routine itself can be as simple or as complex as you like. SUPERMON merely displays the contents of the program counter and accumulator, delays (or stops if TV at $A656=0) and resumes execution of the program. There are, however, a few problems involved with user trace routines.

A user trace routine will be located in RAM or ROM outside the SUPERMON ROM. Therefore, the chip select signal which disables the debugger is ineffective. The SYM solves this problem by allowing the debug switches to be controlled from software. By inserting links W-24 and X-25, VIA#3 at $AC00 can now control the debugger. These links are located just above the speaker.

The initialization routine of your new debugger must put $80C0 in the NMI vector $A67A and the address of the actual trace routine in $A674. When the NMI is raised, SUPERMON will call ACCESS, save the registers, switch off the debugger and jump to your tracer which should then print the registers, memory or whatever, and finish with a -

```
4C CD80          JMP TRACON
```

to switch on the debugger and resume execution. Some condition should be included to halt execution or the debugger will trace forever or until the program returns to SUPERMON.

There are a few interesting points to note with regards to the debugger.

1. DO NOT USE THE 'BRK' INSTRUCTION. This instruction in a user program will interfere with the debugger's operation.
2. The debugger can trace programs in ROM. Many similar systems insert BRK instructions in the user program for single stepping which requires the program to be in RAM. I have successfully single stepped BASIC with the SYM debugger.
3. The chip select disabling capability means that the debugger will not bother single stepping SUPERMON subroutine calls.
4. The NMI input to the 6502 can not be used for any other purpose without hardware modifications.

Try a simple trace routine at first to get the feel for the debuggers operation before attempting a major trace program.

NEXT MONTH - THE SPEAKER

In the final SYM-POSIUM article, we will look at the small piezo-electric speaker on the SYM and how it can be used to play music or generate sound effects. We will also look at other methods of sound generation.

Brian Campbell

THE MEETING WAS KAOS

RABBLE EXPANSION BOARD: Bill Chilcott had his new board on show at the meeting. Details on availablity and price are listed elsewhere in this newsletter.

65C02: Rockwell has released their new 65C02 5 MHz CMos microprocessor. The new chip will contain an extra 27 instructions over the Mos-Tech 6502. Rockwell is under licence to Mos-Tech.

SOFT FRONT PANEL: A part of the GTBUG package written by Tony Durrant is his new soft front panel. This new feature will single step a program, change memory, disassemble instructions into mnomonics and breakpoint can also be set. Tony will not release GTBUG for sale untill he is completely satisfied with it, so I'm afraid we will all have to patient for a while longer.

FORTH: mention of a new local newsletter based around Forth is in the pipeline and should interest most Forth users.

TAPE LIBRARY: John Whitehead, known for his numerous articles in the KAOS newsletters, has volunteered to run the Cassette Library from now on. John's phone number is 763 5983.

Those were the main points of the meeting though much more information is exchanged during the ragchew before and after the meeting.

73's from Melbourne

Rod Drysdale VK3BYU

****

SPECIAL DEALS FOR KAOS MEMBERS FROM COMP- SOFT

Rabble Expansion Board	Board only	$85.00
	Full Kit	$305.00
	Assem. & Tested	$380.00

COMING SOON: BMC High Resolution Green Screen, 15MHz band width $195.00

Microline 80 Printer	still avaliable	at	$535.00
2732 EPROM	good price		$5.50

Special price for 1 month till the 30th September.

TASAN VIDEO BOARD	64X32 screen format	Board only	$40.00
		Partial Kit	$115.00
		Full Kit	$130.00

RABBLE EXPANSION BOARD

The Rabble Ozi Expansion Board is designed to suit Superboard 11 and C1P computer cases. It provides for 32K of 6116LP3 RAM, addressed at locations $0000 - $7FFF. Facilities are also provided for a further 32K of ROM at locations $8000 - $FFFF, using 2732s. The Board is strappable to suit 2532s, 2716s and also 2764s.

The Rabble Ozi Expansion Board has an OHIO compatible Floppy Disk Controller for 5¼" or 8" drives, complete with data separator and motor controller. Other features are a Real Time Clock, a PIA and VIA to provide I/O connections for a parallel printer. Last is the dual Programmable Sound Generators providing 6 channels of sound which can provide a simple noise such as a gunshot or explosion.

The current consumption of this Board is less than 1 Amp at 5V (current varies with loads on I/Os).

Available from:

Rabble Ozi Computers	COMP-SOFT	Data Parts
P.O. Box 781	235 Swan St	5 Naomi St
Shepparton Vic. 3630	Richmond Vic. 3121	Shepparton Vic. 3630

65U DOS RAMBLINGS
by David Tasker

A few months ago I obtained a copy of 65U for my much modified C1. After conferring with a couple of the more active disk users in KAOS who suggested my time might be better used designing more hardware, I decided to try and unravel some of the differences between the 65D and 65U operating systems.

65U is best understood as a core or kernal based system. i.e. There is a central operating system which is accessed by the high level languages as required. We can send commands to the core from the Extended Monitor, Assembler or Basic. Typically, if we were in Basic and we wanted to load a program we would type: DISK!"LOAD FILNAM" We could then list and modify the program and RUN or SAVE it. If we wanted to SAVE it under a different filename, we would have to CREATE a place for it on the disk with a program called CREATE. 65U would SAVE the program by Track and sector.

65U is primarily a BASIC system. Commands to the DOS for load could be, LOAD"FILNAM" if the original program was read/write access (R/W) then no password is needed. However, we can with 65U specify at the time we CREATEd the program location, asign a 4 character password. e.g. LOAD"FILNAM","PASS",10 where PASS is our password and 10 is the line number to enter the program.

With 65D you might consider that all the DOS commands are a subset of the BASIC word DISK!" where as in 65U the DOS commands are a part of BASIC. Another example. Say we currently booted our system in from Drive A. (This would normally be the case.) Part way through our program we wanted to get a DATA file from the Disk loaded in Drive B. The syntax for 65D and 65U including line numbers is shown to compare.

65D	65U
10 DISK!"SE B"	10 DEV"B"
20 DISK OPEN,6, "FILNAM"	20 OPEN "FILNAM",1

The 6 in the 65U is a buffer in memory called device 6. Anything in this BUFFER can be inputed into the program (INPUT 6) and we can print to this buffer (PRINT 6). There are two buffers that we can use (6 and 7).

With 65U the 1 indicated a channel (channel 1). We can input (INPUT%1) and print (PRINT%1) also CLOSE device or channel number can be used. This will cause any new or changed data to be written back to the disk file. In 65U you can have up to 8 channels opened at any one time.

65U does not have a kernal or core that you access. There is no Extended Monitor or Assembler. If you want to access Assembler or 65D files, then 65U does have a program called LOAD32 (for 32K systems) and LOAD48 (for 48K systems) which gives you a kernal like operating system for accessing 65D files.

65U also requires more operating memory. The DOS is 24K long and needs 8K of work-space. So a minimal system is 32K. Quite a few very powerful BASIC keywords for file handling.

Continued next month

****

FOR SALE

AIM 65 with 12K of Assembler Monitor in ROM, 36K memory, Digital Cassette Recorder, EPROM programmer, 60 I/O lines, Printer on board. All one integrated unit, in one case with power supply. Used for commercial development. $590.00 or ono
Contact Grahame Younghusband

SUPERBOARD II, 8K RAM, DABUG III, 300 and 600 baud, 1-2 MHz, Custom made case, Modulator, Power Supply, Manuals - standard and additional (several) Approximately $200 in software, includes Editor/Assembler, Extended Monitor, utilities and many games. REDUCED FOR A QUICK SALE $400.00 or near offer.
Contact Alex Koehler